2025 Spring OOP Final Project Logic Simulator

Table of Contents

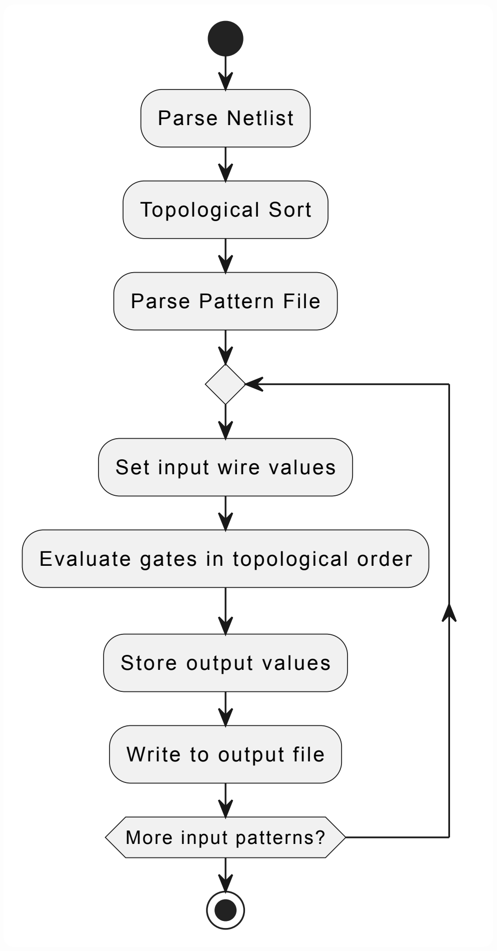
Introduction

A logic simulator is a software program that mimics the behavior of digital logic circuits. It reads a netlist, which is a text-based description of a circuit made up of logic gates and wires, and simulates how signals such as 0, 1, and X (unknown) move through the circuit based on given input patterns.

Problem Formulation

Develop a logic simulator that reads a flattened, hierarchy-free Verilog netlist (.v) consisting of primitive gates (and, or, nand, nor, xor, xnor, buf, not), each with at most two inputs, as well as wire declarations and constant values (1'b1, 1'b0), and simulates the circuit's behavior for each input pattern provided in a pattern file (.pat) containing values 0, 1, and X (unknown).

Flowchart



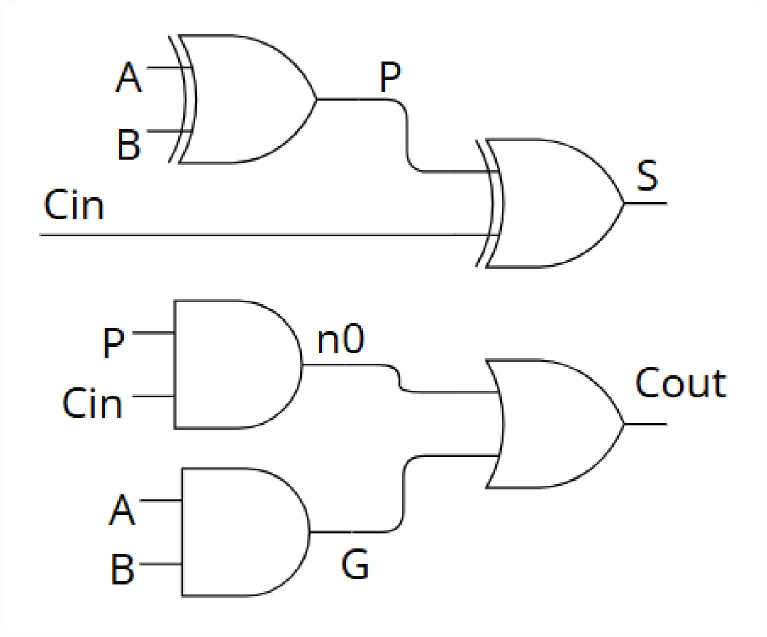
Input Format

Example netlist:

```verilog
module FA (A, B, Cin, S, Cout);
    input A;
    input B;
    input Cin;
    output S;
    output Cout;

    wire P, G, n0;

    xor (P, A, B);
    and (G, A, B);
    xor (S, P, Cin);
    and (n0, P, Cin);
    or  (Cout, n0, G);
endmodule
```



Example pattern:

```
input A, B, Cin
    0 1 0
    1 0 1
    1 X 1
    1 1 0
.end
```

Output Format

Example output:

```
output S, Cout
    1 0
    0 1
    X X
    0 1
.end
```

Evaluation

1. Verilog netlist parsing
    - Accurately parses primitive gates.
    - Correctly handles wire declarations and constant values.
2. Simulation engine
    - Implements correct logic for both 1-input and 2-input gates.
    - Properly propagates unknown values.
    - Ensures correct wire value updates using *topological sorting.
3. Object-oriented design (inheritance & polymorphism)
    - Defines an abstract base class (e.g. `gate`) with at least one pure virtual function (e.g. `evaluate()`).
    - Implements each gate type (e.g. `buf_gate`, `not_gate`, etc.) as a subclass that overrides the function.
    - Uses dynamic dispatch to evaluate gates polymorphically during simulation.
    - Includes proper use of virtual destructors to avoid memory issues.
4. Executable usage:

    ```
    ./ls -i *.v -p *.pat -o *.out
    ```

    e.g.

    ```
    ./ls -i FA.v -p FA.pat -o FA.out
    ```

Grading

- Functionality: 60% (all specs in Evaluation are met)
- Performance (runtime): 30%
- Report: 10%

Naming error: -10 points

Bounses

- (5 points) The logic simulator will automatically detect combinational loops and multi-driver nets in the netlist.
- (10 points) Generates functionally equivalent netlists with reduced gate counts based on the given Verilog file, including dff modules defined as follows:

```verilog
module dff (
    input clk,
    input rst_n,
    input d,
    output reg q
);
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            q <= 1'b0;
        else
            q <= d;
    end
endmodule
```

Report

Please write a short report explaining how your program works. Your report should include the following:

1. A brief overview of your project and what it does.
2. A description of how and where you used inheritance and polymorphism in the project.
3. Any bugs you encountered during development and how you resolved them.
4. Special features or bonus functionalities you added.
5. A summary of how you divided the work among team members, including each person's contribution percentage.
6. Your feedback on the project and a short conclusion.

Submission

1. File name:
   - Your final file must be named: `student_ID.tar` (e.g. `113511000_113511999.tar`)
2. Inside the .tar file:
   It should contain a folder named `student_ID`, which has:
   - Source code files (`.h`, `.cpp`)
   - Makefile
   - Report named `student_ID.pdf`

3. How to make the .tar file:

```
tar cvf student_ID.tar student_ID
```

4. Submit to the New E3 before the deadline.

Q&As

Feel free to ask us via email.

Contacts

- TA1: Shin-Kuan, shinkuan318@gmail.com
- TA2: Chong-En, cehong.ee13@nycu.edu.tw
- TA3: Wei-Ting, wendy4741@gmail.com

Please direct all questions or requests to all TAs; messages sent to just one TA may go unanswered. To keep things fair, we'll post every Q&A in the shared discussion thread. We aim to respond within one day, so thanks in advance for your patience.

Note

Plagiarism is strictly prohibited. Code from the Internet cannot be used directly. If found, the score will be 0. We will report to professor Liu and handle the matter according to the relevant regulations.

References

1. Regular expressions library (since C++11)
2. Topological sorting

Truth Tables

AND Gate

A \ B	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

OR Gate

A \ B	0	1	X	Z
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

NAND Gate

A \ B	0	1	X	Z
0	1	1	1	1
1	1	0	X	X
X	1	X	X	X
Z	1	X	X	X

NOR Gate

A \ B	0	1	X	Z
0	1	0	X	X
1	0	0	0	0
X	X	0	X	X
Z	X	0	X	X

XOR Gate

A \ B	0	1	X	Z
0	0	1	X	X
1	1	0	X	X
X	X	X	X	X
Z	X	X	X	X

XNOR Gate

A \ B	0	1	X	Z

0	1	0	X	X
1	0	1	X	X
X	X	X	X	X
Z	X	X	X	X

BUF Gate

A	Y
0	0
1	1
X	X
Z	X

NOT Gate

A	Y
1	0
0	1
X	X
Z	X